CORE
Metadata, citation and similar papers at core.ac.uk
Provided by DigitalCommons@CalPoly

Discrete Alarm Clock

by

Travis Moore and Collin Barth

Senior Project

ELECTRICAL ENGINEERING DEPARTMENT

California Polytechnic State University

San Luis Obispo

Spring 2011

Table of Contents

List of Tables / Figures

Acknowledgements

- Dr. Hugh Smith, Cal Poly
- Dr. John Oliver, Cal Poly
- AVRfreaks.net

Abstract

The standard alarm clock, as used daily by millions worldwide, has clear room for modification and improvement. This paper documents an implementation that rectifies many of these flaws. Improvements include adjustable snooze length, an IR remote for ease of use, and an RF headset for discrete alarm use or potential incorporation into a device much like a hearing aid. The processes involved in creating a digital alarm clock, as well as the obstacles met in the implementation of the enhanced design, are detailed within.

I. Introduction

Our project began with an investigation into the standard household digital alarm clock, an identification of its flaws, and a vision of how it could be improved. This project develops an alarm clock that is easy to customize, easy to use from any location in a room, and practical to use without disturbing another person sleeping in the same room. This ability to use the alarm function discretely is the inspiration for the project.

After brainstorming solutions to the issues with current alarm clocks, we decided upon three main adjustments we could make to produce an improved version of our own. We decided that the addition of a remote control, an optional earpiece, and a customizable snooze time would improve the current state of alarm clocks. Our design incorporates an Infrared (IR) remote able to snooze the alarm as well as turn it off, and a Radio Frequency (RF) portion to simulate an earpiece able to isolate the sounding of the alarm as desired by the user. We created an alarm clock from the ground up using ATmega microcontrollers, in order to add the aforementioned improvements.

This undertaking incorporated the knowledge we have gained from many of our courses here at Cal Poly, while relying on the creative problem solving and other skills that we have developed throughout our experience here. Our programming practice from our CPE classes was relied upon

heavily in order to manipulate the microcontrollers that are running our clock and its accompanying systems. The topics and methods we covered in our circuit design courses were used to produce the functioning RF and IR circuitry in our final clock design. We also followed basic soldering and electrical principles while constructing the housing for our clock. Our main goal in this project was to continue what we started here at Cal Poly on day one, and 'Learn by Doing!'

II. Background

Before designing our project, a list of flaws with current alarm clocks had to be compiled and evaluated. We immediately realized that one aspect of alarm clocks that could be improved was the disturbance created for everyone in the same room as the clock. While that is sometimes the intended result, in a majority of cases an alarm is set to wake up one person and is an inconvenience to others in the same room. This scenario is played out most often in two common situations, couples sleeping in the same bed, and college roommates sleeping in the same room. Another issue we wanted to address was the difficulty of using an alarm clock from anywhere other than within an arm's reach. A solution to this problem would be more of a luxury than a necessary improvement. Finally, we decided to improve upon a feature already available on most alarm clocks, the snooze button. All snooze buttons we have encountered have a predetermined length of time, and it seems that a customizable snooze time would be a desirable option for our clock.

We decided to develop our own functioning alarm clock via ATmega microcontrollers and incorporate buttons, switches, a radiofrequency (RF) transmitter/receiver pair, and an infrared (IR) remote and receiver.

III. Requirements

Discrete Digital Alarm Clock:

- Fully functional digital clock with an alarm function
- RF receiver circuit used to “simulate” a discrete headset or RF hearing aid
- IR remote to control basic functions of the clock, such as putting the alarm to sleep, and snoozing the alarm
- Adjustable Snooze time that can be set between 1 and 30 minutes
- Controls on the clock so that the alarm can sound through either the headset or the speaker on the main clock unit, or through both the headset and the speaker
- Main digital clock unit running from a 120V, 60Hz AC wall socket
- Simulated headset running on a 6V battery pack
- LEDs on the clock screen to display an AM/PM indicator, the state of the alarm, and whether the headset is engaged
- ‘Alarm Set’ button to put the alarm to sleep when it is sounding, or to set the alarm time when used in conjunction with the hour/minute buttons
- ‘Time Set’ button to set the clock time when used in conjunction with the hour/minute buttons

- ‘Snooze’ button to turn on the snooze function when the alarm is sounding, or used in conjunction with the minute button to set the snooze delay
- ‘Hour’ and ‘Minute’ buttons to increase the hours and minutes during time setting functions, incrementing at a rate of around 2Hz
- ‘Alarm On/Off’ switch to control the operating state of the alarm
- ‘Headset On/Off’ switch to control the operating state of the headset

IV. Use Cases

In this section, we discuss a number of use cases that describe the full use of our modified alarm clock.

1) User desires to use the product as a standard clock with no alarm function. Plug the unit into a wall outlet. Press and hold the **'Time Set'** button while adjusting the time with the **'Hour'** and **'Minute'** buttons. Ensure **'Alarm'** and **'Earpiece'** switches are in the 'off' position.
2) User desires to use the product as an alarm clock, without the headset, the remote, or the snooze function. Plug the unit into a wall outlet. Press and hold the **'Time Set'** button while adjusting the time with the **'Hour'** and **'Minute'** buttons. Press and hold the **'Alarm Set'** button and adjust the displayed time to match the desired alarm time with the **'Hour'** and **'Minute'** buttons. Ensure **'Alarm'** switch is in the 'on' position and **'Headset'** switch is in the 'off' position. When alarm sounds, press **'Alarm Set'** to put alarm to sleep until alarm time next day. Alarm will only sound on the main clock speaker.
3) User desires to use the product as an alarm clock with a snooze function, without the headset, or the remote. Operate unit as described in (2). After setting alarm time, press and hold **'Snooze'**

button while pressing **'Minute'** until desired snooze length appears on screen. When alarm sounds, press **'Snooze'** to delay sounding of alarm by set time or press **'Alarm Set'** to put alarm to sleep until alarm time next day.

4) User desires to use the product as an alarm clock as previously described but with the headset sounding and not the main speaker. Operate unit as described in cases (2)-(3), but ensure **'Alarm'** switch is in the 'off' position and **'Headset'** switch is in the 'on' position. The alarm will sound only in the RF headset.

5) User desires to use the product as an alarm clock with previously stated functions with both the headset and the main speaker. Operate unit as described in cases (2)-(3), but ensure **'Alarm'** switch is in the 'on' position and **'Headset'** switch is in the 'on' position. The alarm will sound in both the RF headset and the main clock speaker.

6) User desires to use the product as an alarm clock with any of the previously stated functions and the IR remote as well. Operate unit as described in cases (2)-(5). IR remote can be used to snooze the alarm or put it to sleep. Remote button **'1'** will put the alarm to sleep if it is pressed while the alarm is sounding. Remote button **'2'** will snooze the alarm for the set time if pressed while the alarm is sounding.

V. Project Design

Our design consists of three breakout boards, each of which contains circuitry and a microcontroller that contribute different functions to the Discrete Digital Alarm clock. An AVR ATmega32 microcontroller is mounted on each board to give us complete control of the system's operations. Two of the boards are located in the main clock unit, and the third is used to simulate the RF headset. In the main clock unit, one microcontroller acts as the central hub of the entire system. This board will be solely responsible for controlling and multiplexing the display, keeping track of the time and alarm, sending out RF signals to the headset, receiving button and switch inputs, and sending signals between the other microcontroller located in the main clock unit. The second microcontroller in the main unit is used to receive IR signals from the remote, and control the speaker in the alarm clock. Signals sent between these two microcontrollers coordinate certain tasks to keep the clock running as one unit, such as the sounding of the alarm and the actions controlled by the IR remote. Using two microcontrollers on this end was necessary due to the number of I/O pins needed for the 7-segment clock display and the various buttons and switches controlling our device. Splitting up the processing due to the limited I/O pins on the ATmega32 microcontrollers turned out to work in our favor, as we were able to keep

operations separate that consume large portions of available power on each chip. The third microcontroller is used to simulate the RF headset, and only handles the circuitry dealing with the RF receiver and the buzzer to sound the alarm.

The main clock unit runs on a 120V, 60Hz AC supply, and the simulated RF headset is powered by a 6V battery pack. These power options provide a realistic scenario as most digital alarm clocks are powered from the wall outlet, and many small hearing devices are powered off small coin cell batteries than can range anywhere between 1V and 6V.

User input to the Discrete Digital Alarm Clock will come from the use of 5 buttons and 2 switches on the main clock unit or from the 2 buttons on the IR remote. The functions controlled on the main unit are outlined in Table I.

Table I: MAIN UNIT INPUTS AND FUNCTIONS

Button(s) pressed	**Function**
Alarm Set	Alarm will be turned off if it is sounding, otherwise Alarm Time will be displayed
Snooze	Alarm will be snoozed if it is sounding, otherwise Snooze length will be displayed
Hour	No function if used alone
Minute	No function if used alone

Time Set	No function if used alone
Time Set + Hour	The current clock time will be increased by one hour at 2Hz
Time Set + Minute	The current time will be increased by one minute at 2Hz
Snooze + Minute	The snooze delay will be increased by one minute at 2Hz
Snooze + Hour	Hour has no function when pressed with snooze; snooze time will be displayed as usual.
Alarm Set + Hour	The alarm time will be increased by one hour at 2Hz
Alarm Set + Minute	The alarm time will be increased by one minute at 2Hz

In addition to the switches and buttons on the main clock unit, an IR remote can be used to control some of the functionality of the Discrete Digital Alarm Clock. Table II outlines the usage of the buttons on the IR Remote.

Table II: IR REMOTE BUTTON FUNCTIONS

Button Pressed	**Function**
1	Put the alarm to sleep when sounding
2	Snooze the alarm when sounding

VI. Design Plans

Implement the clock display: One of the first steps in building this project was to reverse engineer the 7-segment display taken from a commercial digital alarm clock. We used the display from an existing clock because of difficulty finding individual clock display large enough to see clearly from any distance greater than a couple feet, which is a necessary feature in an alarm clock display. In order to reverse engineer the display, we had to figure out what segment each of the 18 pins on the display header controlled, depending on which of two reference pins is supplying voltage at that time. We then had to figure out how each of the four digits was multiplexed to display the time. **Appendix D** shows the circuitry connecting the clock display to the microcontroller.

Create a function to accurately increment the clock: One of the most important issues with creating a digital alarm clock is creating an accurate clock. In order to do this, we used the internal oscillator on an ATmega32 microcontroller and a timer-overflow based interrupt to increment our clock after a calculated number of clock cycles. Since this interrupt is the priority our clock stays accurate to the micro second.

Keep track of the time and display it on the screen: Once the timer was implemented, we could implement a method to keep track of the time.

We used three different eight bit values to keep track of the seconds, minutes, and hours of the current time. Each of these values will roll over when they reach their maximum values (seconds and minutes at 59 and hours at 12). Once the seconds reach a value of 60, they are reset to 0 and the minutes are increased, the same idea is used with the minutes affecting the hours. When the hours reach a value of 12 they are reset to 1 and the AM/PM flag is toggled. The initial startup time is 12:00:00 AM as with most manufactured digital alarm clocks.

Send signals between the RF transmitter and receiver

One of the biggest parts of this project was the RF communication between the main clock unit and the simulated RF headset. We started by using the UART capabilities of the microcontrollers to both transmit and process the data we were trying to send. In the lab we were able to examine the waveforms on both the sending and receiving ends to ensure that the proper data was being transmit. While our sent and received data was nearly identical, we could not overcome the noise and timing issues that arose and chose to eventually abandon using UART transmission because of these setbacks. We created our own bit patterns to transmit and receive and eliminated error with complex bit checking sequences. **Appendix D** shows the circuitry connecting the RF receiver to the microcontroller.

Send signals between IR transmitter and receiver

When we first started the IR portion of the project, we hooked up the IR receiver to the microcontroller and used a TV remote control to check if we were properly receiving the signals. We then created some circuitry to drive the IR LED we planned on using to generate our IR signal. We quickly realized (as explained later) that the IR remote was not possible using the hardware we had because the IR LED did not have the range we needed. We ended up using a four button remote and decoding the signals using an oscilloscope and the IR receiver. We used a microcontroller to analyze the incoming IR signals and distinguish between the buttons being pressed. **Appendix D** shows the circuitry connecting the IR receiver with the microcontroller.

Put all of the working pieces together

At this point in the project we had all of the necessary parts of our project working separately so it was time to combine them. This was the most difficult part of the project due to the complicated timing issues associated with the different components. Separately, the time was based only on the necessary functions running on each microcontroller. When combined, the addition of other system's interrupt service routines changed the timing used to calibrate the IR and RF transmissions. Multiple weeks of work testing and tweaking our timing was necessary to have everything configured and working together, and it took until the deadline of our project window.

Implement all the buttons on the clock

With all of the wireless components of the project completed, the hardware could be added to the main clock unit to provide the necessary user interaction. We used a total of five buttons and two switches to make the clock functional. We had to test and implement another timer and interrupt service routine in order to increase the hours and minutes at a steady pace while adjusting the clock time, alarm time, and snooze delay. The switches are used to control the on or off state of the main clock's alarm and of the headset's alarm.

Implement the alarm function

With all of the basic components together, buttons and switches could be used to implement the alarm functionality of the clock. This stage was tackled last because the RF and IR signals could be used to control many of the functions in the alarm during testing. Once the alarm goes off, the RF signals to turn on the headset are transmitted if the headset is to be used. The IR remote can be used to snooze or turn off the alarm.

VII. Development and Construction

Initial Step: The first step in starting this project was to familiarize ourselves with the AVR ATmega microcontrollers and AVR Studio 4 which was used to program them. Our initial test consisted of turning LEDs on a breadboard on and off to ensure we understood proper timing and how to set specific pins as inputs and outputs. Our development plan was to create separate circuits on breadboards for the three main sections of our overall digital clock; the clock display, the RF transmitter/receiver, and the IR receiver. Three different microcontrollers were used to separately control and test the circuits set up on each breadboard, each programmed from a separate project file in AVR Studio.

Designing the Display: The first circuit we started building was the 7-segment display control; utilizing a display screen from a professionally manufactured clock. Using the results we obtained from our initial testing of the display, we were able to crudely induce different digits to display through use of the ATmega32. This process lead to the realization that the display would require much more circuitry than originally envisioned because of the amount of current needed to drive all of the segments. We planned to use NPN transistors to ensure the cathodes of our test LEDs were being driven to ground and to use the microcontroller to provide the current necessary to

drive the display. The display requires much more current than the ATmega32 can provide though, so a high current Op-Amp was inserted as a voltage follower to provide the necessary driving force.

The Clock Module: A necessity of this project is an accurate timer to control the clock. We were able to implement one using the internal oscillators on the ATmega32 microcontroller and internal interrupts. The clock timer was setup to increment the seconds as the highest priority interrupt on the chip using the most reliable time keeping method available with our hardware. An interrupt service routine (ISR) is triggered once a second using timer counter overflows. A separate set of functions was designed to increase the minutes and hours, while keeping track of the current time.

Buttons and Switches: In order to utilize the alarm and snooze functions of the clock a user interface is necessary. We selected buttons and switches as our medium, along with our IR remote. In the Discrete Digital Alarm Clock, buttons are used to set the clock time, alarm time, snooze delay, turn off the alarm, and snooze the alarm. To make the clock time, alarm time, and snooze delay reasonably adjustable, the display must show the item being updated and automatically increase the time for ease of use. This required another timer on the microcontroller be used to allow for a manageable speed of continuous adjustment of the hours and minutes for the clock time, alarm time, and snooze delay. The timer is setup to increase at

half-second intervals, and the display value increments every time the timer count increases.

The Alarm Functionality: The alarm function was integrated into the existing clock module once it was accurately keeping track of time. Once a user has set the alarm time, the function is used to compare the current time with the alarm time, and a flag is set when they are equal. This flag is used to sound the alarm or signal the RF transmissions to the headset.

The Snooze Functionality: The snooze function was implemented only after there was a working alarm system. After the snooze delay is set and the snooze is activated, a function quiets the alarm and delays the alarm time by the specified amount. The snooze function can be used as many times as desired before the alarm is put to sleep with the alarm set button.

RF Transmitter and Receiver: The RF Transmitter and Receiver are used to communicate between the main clock and the headset. This is to signal to the headset to sound when the alarm goes off. Initial plans were to use UART to communicate between the microcontrollers and the RF instruments. This approach was abandoned due to issues with noise filled data lines as discussed in the Problems section. Our final implementation of RF communication consists of manually sending out bits on the transmitting side in a pattern of our choice. A series of manual checks is made for each bit on the receiving side to identify and interpret our signals. This allowed for a more accurate transmission of data and ensured the integrity of the data

being sent by avoiding noise and interference from other sources. Using this method we are able to consistently turn the RF headset on and off in accordance with the alarm's activity.

IR Remote and Receiver: An IR Remote can also be used to snooze or put the alarm to sleep. The first step of implementing the IR portion of this project was to decipher the IR codes coming from the remote. Once we understood the codes from the remote, we setup test cases on the receiving side to interpret the signals. A function was created to test the stream of bits coming from the IR Receiver. When the incoming sequence does not match the codes the function returns in preparation for the next sequence. The function returns a different value depending on which valid code was received, and this information is used to set flags for later processing.

System Integration: Once all the separate components were working in a standalone configuration, they were integrated together. The first step of integration was synchronizing the alarm functionality with the RF transmitter and receiver. When the alarm is triggered, a RF transmitter sends the sound signal to the receiver if the headset feature is turned on. The second step in integration was setting up the IR remote to signal functions on the alarm. Signals were added to the main clock microcontroller from the IR side in order to trigger events that the microcontroller would process in coordination with buttons and switches on the clock. We run the speaker for the main alarm clock off the IR microcontroller in order to avoid drawing large currents from

our main microcontroller, because that interference was affecting our clock's function and the RF data we transmit.

Assembly: Once the entire system was working as designed, final assembly began. The first step of this was to solder all of the components onto soldered breadboards for a permanent configuration. One board was used for the simulated RF headset, and two were used in the main clock unit. Soldering the components to breadboards ensured that there would be no lose connections during operation, and made the unit much more portable. One of the last steps in the assembly was to create the clock unit where the circuits would reside and provide a place to mount the display, buttons, and switches for the discrete digital alarm clock. We fashioned a body for our clock out of wood that we purchased, cut, and stained. We included a Plexiglas cover for the box to show the circuitry inside the clock. As a final step, the circuits were secured inside the clock unit and the buttons and switches were mounted.

VIII. Problems

Driving the 7-Segment Display: One of the first major problems we encountered occurred while attempting to drive the clock display that we reverse engineered. Our initial thinking was that the ATmega32 microcontroller could drive each of the cathodes to ground and two pins could provide current for the two anodes of the display, because this is what was needed to run the display. We quickly learned that the display required much more current than we initially thought. The two anodes of the display required over 40mA of current, much more than the microcontroller could supply, so we had to find another solution. We devised a simple circuit to supply current by using two NPN transistors to drive two high current Op-Amps that would supply the current and voltage necessary for the display. Once we implemented this circuit into our design, we realized that the cathodes of the displays were not being driven to ground because again the microcontroller could not sink the amount of current needed. In order to remedy this situation, we used NPN transistors on each of the cathodes that had their emitters connected to ground, and the collectors wired to the cathodes. The microcontroller could now easily turn on each of the transistors and properly drive each of the LEDs to ground so they could turn on when needed.

RF Transmitter and Receiver: The RF Transmitter and Receiver caused us the most trouble in this project. The hardware that we used was extremely limited in its noise filtering and required us to take a different method than we initially anticipated. We started this part of the project by using the UART transmission on both the transmitting and receiving side, but quickly had to abandon that approach. The UART transmitter is designed to quickly transmit one byte of data by storing the byte in the data register, placing it into a shift register after space is made, and sending it out one bit at a time. This worked fine on the transmitting side, but on the receiving side excess noise ruined the data collection. On the detection of a high to low transition, the receiving UART side immediately starts reading in data, bit by bit, until it has filled up the data register. When the receiver detected a high to low transition in the noise, it would automatically begin collecting data, causing our functions to start collecting data before the actual start bytes were being sent. Because of this, our data was often invalid causing our error detection scheme to ignore it.

Our final solution to this problem, after various attempts at modifying the data to prevent interference from noise, was to ‘bit-bang’ the data interaction on both the transmitting and receiving sides. Bit-banging allowed us to manually control what data we sent and received, bit by bit, instead of a byte at a time. By looking at both the transmitting and receiving signals, we could create signals that would be less affected by noise, and create our own

error checking scheme that way. Although these signals were still affected by noise, we were able to slow down our bit rate and effectively cancel out the noise on our signal to ensure that we collected valid data.

Timing Issues: We encountered a number of different timing issues on the main microcontroller. This chip is in charge of the timing for the clock display, RF transmission, and IR reception. Our main priority was the timer used for the keeping track of time because of the high importance of an accurate clock. However, because this took the highest priority, many of the other timers and controls that were running were skewed when this interrupt service routine executed. We had to ensure that other tasks would not be affected by this. We ran different timers to control the various functions of the clock allowed, which allowed us to multiplex the display at the same time the clock was running without problems. In addition, we moved the IR receiver to another microcontroller to avoid constant errors in our reception of the IR signal. We also moved the responsibility of sounding the buzzer for the alarm to this supplementary microcontroller to ensure that the current needed to activate the speaker would not interfere with the main display of our clock. To allow these microcontrollers to communicate with one another, we put in two output lines from the IR receiving microcontroller to the main microcontroller that multiplexed the various remote commands. In addition, a notification signal from the main microcontroller notified the IR receiving side when the

alarm should be on or off so that the buzzer could be turned on or off accordingly.

Breakout Board Malfunction: Our final problem with the implementation of our design occurred on the day before we were to present our finished product. The breakout board be used for our RF microcontroller had apparently developed a short or a bad connection somewhere near the ATmega chip, causing the chip to become useless. We also fried a backup chip we attempted to program in the same DIP socket. Luckily, we had two extra microcontrollers and were able to salvage our project by placing the RF circuitry on a breadboard and avoiding the suspect breakout board all together.

IX. Conclusion

Travis Moore: This project was designed to show how an everyday alarm clock could be turned into a discrete alarm clock using technology readily available. For example, a hearing aid could be turned into a RF headset for a comfortable alarm for someone who already wears one because they are hearing impaired or for someone who wishes to wake up without disturbing others nearby. Throughout the course of this project we incorporated different ideas and knowledge obtained through classes at Cal Poly. We used microcontrollers to run the tasks needed to create a functioning digital alarm clock, created and analyzed circuits used, and examined different methods of wireless communication. This project effectively utilized the skills I have learned as an Electrical Engineer and put them to use in a project that can easily be expanded and marketed. Throughout the process of doing this project I encountered many problems I did not realized existed and discovered methods to overcome then; a skill that will always be useful later in life. Working on this project opened my mind to other ideas and projects that I can do, many of which will involve and be concentrated on things learned while building this Discrete Digital Alarm Clock.

Collin Barth: Entering Cal Poly as a freshman I was very excited to find out what I would 'Learn by Doing' in my time at this school. I spent most of my first three years here as a Computer Engineering student, and while I did have a number of labs accompanying my classes, I didn't get to experience as many hands on projects as I would have liked. Since switching into the EE course path and experiencing the circuits classes and their labs I have gained valuable experience working with circuits and generating the expected results. The hundreds of hours of programming practice I got in my CPE classes proved to be very valuable in this project, as well as the practice analyzing and developing circuits in my EE classes. This project showed me that just by applying the learning I have done at Cal Poly to problems I encounter in the rest of my life, I can generate remarkable results. We experienced quite a bit of difficulty bringing this whole project together, especially in the final week. I definitely gained a better perspective of how to approach large tasks in my future career, and a confidence when it comes to diagnosing and resolving issues in my projects, as long as I have allotted enough time to address them. This project will prove to be an invaluable experience and a great stepping stone into the electrical engineering field.

X. Bibliography

"AVR Timers - An Introduction. | EXtreme Electronics." *EXtreme Electronics*. 18 Sept. 2008. Web. 02 June 2011. <http://extremeelectronics.co.in/avr-tutorials/avr-timers-an-introduction/>.

Hewes, John. "Transistor Circuits." *The Electronics Club*. 2010. Web. 20 May 2011. <http://www.kpsec.freeuk.com/trancirc.htm>.

"RF Receiver Help." *AVR Freaks*. 5 Jan. 2011. Web. 17 May 2011. <http://www.avrfreaks.net/index.php?name=PNphpBB2>.

XI. Appendices

A. Specifications

RF Link Transmitter - 315MHz

http://www.sparkfun.com/datasheets/Wireless/General/MO-SAWR.pdf

RF Link 2400bps Receiver – 434 MHz

http://www.sparkfun.com/datasheets/Wireless/General/MO-RXLC-A%20data%20sheet.pdf

IR Receiver Breakout

http://www.sparkfun.com/datasheets/Sensors/Infrared/tsop853.pdf

AVR ATmega32

http://www.atmel.com/dyn/resources/prod_documents/doc2503.pdf

AVR ATmega164P

http://www.atmel.com/dyn/resources/prod_documents/doc8011.pdf

B. Parts List and Cost

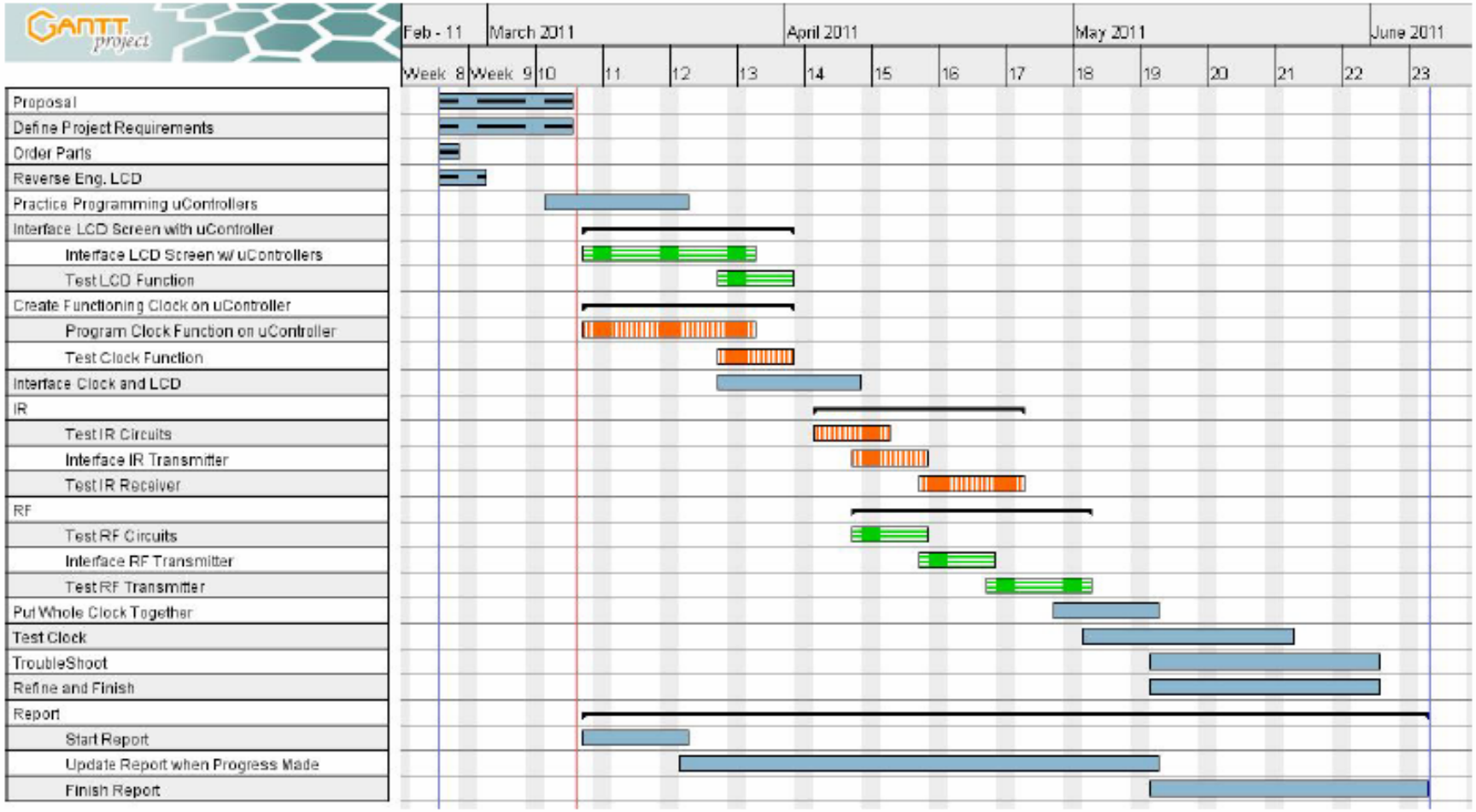
Part	Model Number	Quantity	Per Unit Cost	Cost
Microcontrollers	ATMega32A	3	$4.50	$13.50
Soldered breadboards	n/a	3	$2.50	$7.50
RF Transmitter	WRL-08946	1	$3.95	$3.95
RF Receiver	WRL-08949	1	$4.95	$4.95
IR Receiver	SEN-08554	1	$9.95	$9.95
Resistors 10kΩ		18	$0.10	$1.80
Resistors 270kΩ		2	$0.10	$0.20
40 pin DIP sockets		3	$0.90	$2.70
NPN Transistors	BC184C	18	$0.20	$3.60
Op-Amp		1	$3.00	$3.00
Power JET		2	$1.10	$2.20
DAC	DAC121S101	1	$11.00	$11.00
DC Buzzer		1	$1.50	$1.50
Speaker		1	$1.50	$1.50
7-Segment Display		1	$8.00	$8.00
Push buttons		5	$2.50	$12.50
Various wires		1	$6.00	$6.00
Wood panel		1	$4.65	$3.65
Plexiglas Sheet		1	$2.25	$2.25

Total Cost	**$99.75**

Digital Alarm Clock
2/24/11-6/7/11

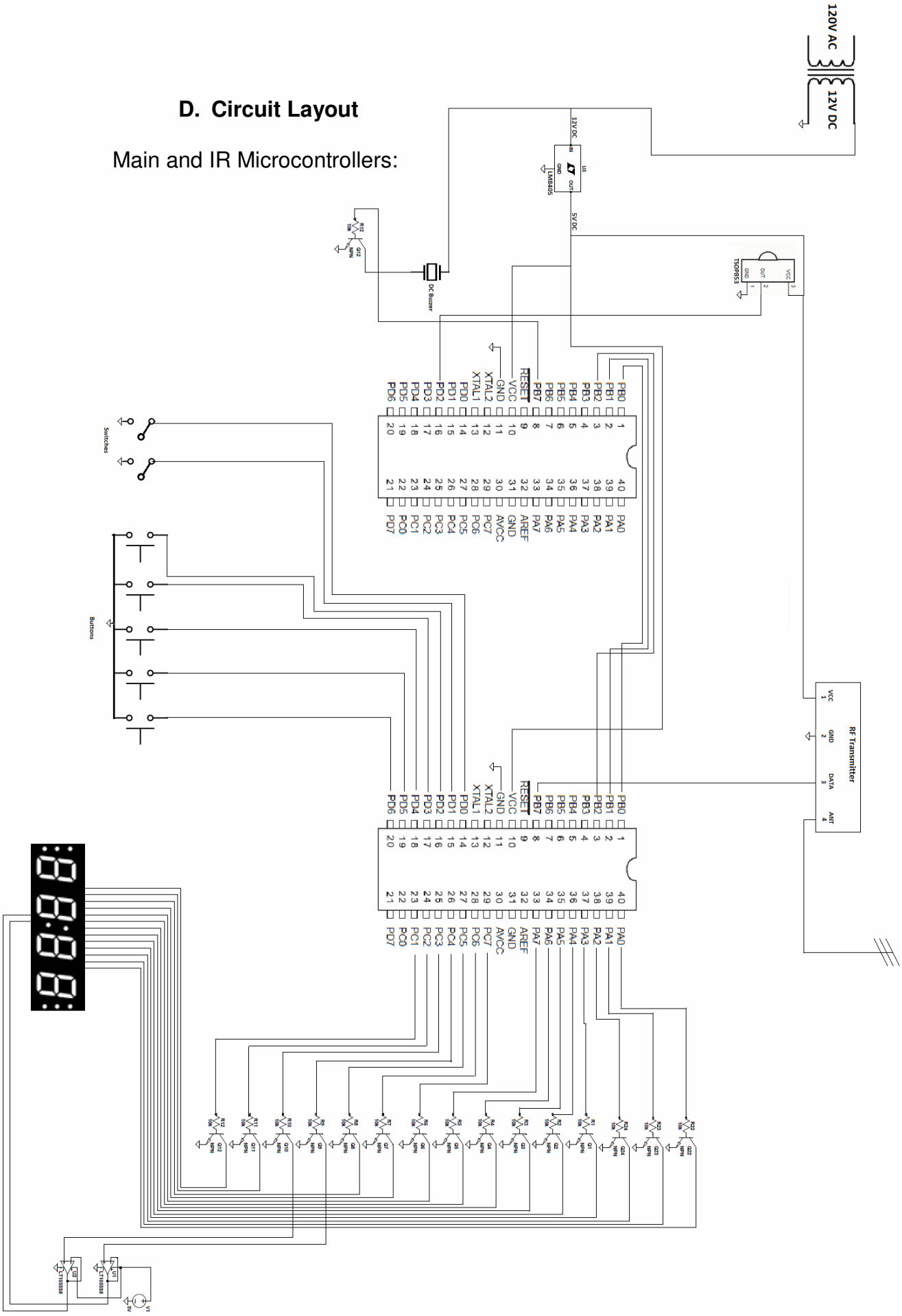
Gantt Chart

C. Schedule – Time Estimates

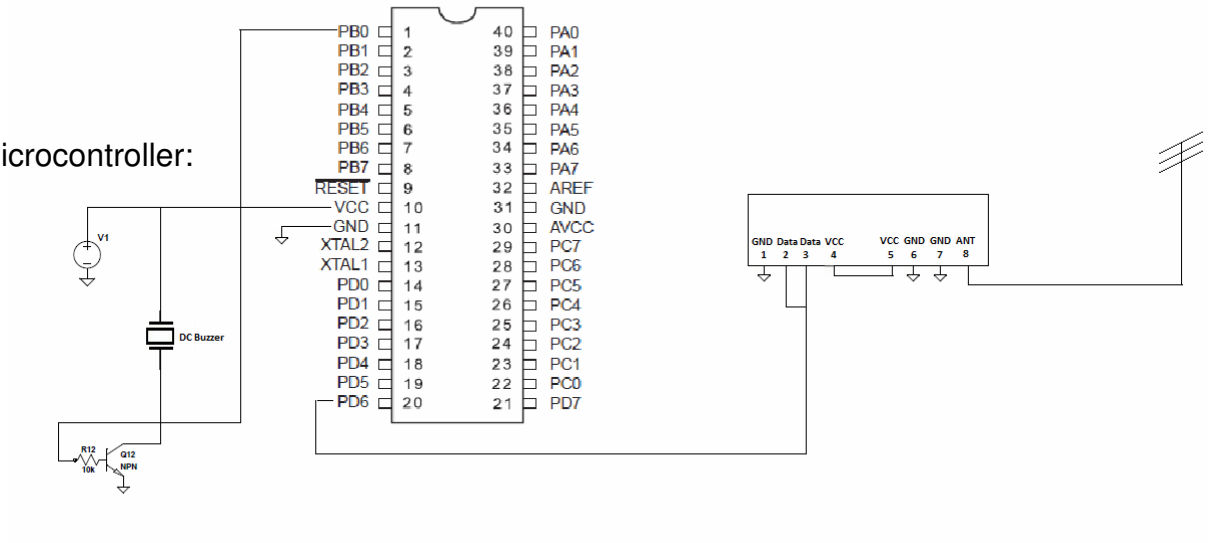


D. Circuit Layout

Main and IR Microcontrollers:



RF Microcontroller:



E. Project Code for AVR Studio

The C code for each of our project files and our two header files begins on the following page.

```
/*************************************************************
Travis Moore and Collin Barth

*The Discrete Digital Alarm Clock*
Electrical Engineering Senior Project
        Spring 2011
*************************************************************/

/*************************************************************
*GlobalInclude.h*
This file contains all the declarations used in this project
*************************************************************/
extern unsigned char b_AlarmSounding; //Alarm is sounding if a 1
extern unsigned char b_Snooze; //Alarm should be snoozed if a 1
extern unsigned char b_AlarmOn; //Alarm is activate if a 1
extern unsigned char b_HeadsetOn; //Headset is activate if a 1
extern unsigned char b_SetAlarm;//If the alarm is being set
extern unsigned char b_SetSnooze; //If the snooze is being set
extern unsigned char Increase_Sec;
extern unsigned char Seconds;
extern unsigned char Minutes;
extern unsigned char Hours;
extern unsigned char Alarm_Minutes;
extern unsigned char Alarm_Hours;
extern unsigned char Snooze_Alarm_Seconds;
extern unsigned char Snooze_Alarm_Minutes;
extern unsigned char Snooze_Alarm_Hours;
extern unsigned char Snooze_Minutes;
extern unsigned char b_Clock_AM; //Set to a 1 if the clock is AM
extern unsigned char b_Alarm_AM; //Set to a 1 if the clock is AM
extern unsigned char b_Snooze_AM; //Set to a 1 if the clock is AM
void Add_Snooze(void);

/*************************************************************
Travis Moore and Collin Barth

*The Discrete Digital Alarm Clock*
Electrical Engineering Senior Project
        Spring 2011
*************************************************************/

/*************************************************************
*BitMask.h*
This file contains I/O functions for the ATmega microncontrollers
*************************************************************/

// Copyright 2009, Tony Givargis.

#ifndef __avr__
#define __avr__

#include <avr/interrupt.h>
#include <avr/pgmspace.h>
#include <avr/io.h>

#define XTAL_FRQ                                8000000lu

#define SET_BIT(p,i)                            ((p) |=  (1 << (i)))
#define CLR_BIT(p,i)                            ((p) &= ~(1 << (i)))
#define GET_BIT(p,i)                            ((p) &   (1 << (i)))

void ini_avr(void);
void wait_avr(unsigned short msec);

#endif
```

```
/*************************************************************
Travis Moore and Collin Barth

*The Discrete Digital Alarm Clock*
Electrical Engineering Senior Project
       Spring 2011
*************************************************************/

/*************************************************************
*Digital_Clock.c*
This file controls all the main functions of the clock
       It initializes all the timers, I/O ports, interrupts, etc
       It checks the status of the buttons, and sends and receives
       signals
*************************************************************/

#include <GlobalInclude.h>
#include "BitMask.h"

#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>

void Send_Packet(uint8_t cmd);

void USART_Init(void);
void Init_Timers(void);
void Init_IO(void);

void Increase_Mins(void);
void Increase_Hrs(void);
void Increase_Alarm_Mins(void);
void Increase_Alarm_Hrs(void);
void Increase_Snooze_Mins(void);

void Clear_Alarm(void);
void Clear_Snooze(void);
void Display_Alarm(void);
void Display_Snooze(void);

int Check_Alarm(void);

unsigned char Message_Sent = 1;

void RF_Transmit(uint8_t data);

//Define all global variables
unsigned char b_SetAlarm = 0;
unsigned char b_AlarmSounding = 0; //Alarm is sounding if a 1
unsigned char b_AlarmOn = 0; //Alarm is activate if a 1
unsigned char b_HeadsetOn = 0; //Alarm is activate if a 1
unsigned char b_Snooze = 0; //Alarm should be snoozed if a 1

unsigned char b_SetAlarm;// = 0; //If the alarm is being set
unsigned char b_SetSnooze = 0; //If the snooze is being set

unsigned char Increase_Sec = 0;
unsigned char Seconds = 0;
unsigned char Minutes = 0;
unsigned char Hours = 12;

unsigned char Alarm_Minutes = 0;
unsigned char Alarm_Hours = 1;

unsigned char Snooze_Alarm_Seconds = 0;
```

```
unsigned char Snooze_Minutes = 5;
unsigned char Snooze_Alarm_Minutes = 0;
unsigned char Snooze_Alarm_Hours = 0;

unsigned char b_Clock_AM = 0;
unsigned char b_Alarm_AM = 0;
unsigned char b_Snooze_AM = 0;

int main(void)
{
	Init_Timers();
	Init_IO();
	//Enable global interrupts
	sei();

	while(1)
	{

		if( (b_AlarmOn == 1) || (b_HeadsetOn == 1) )
		{
			if(Check_Alarm())
			{
				//For the buzzer
				if(b_AlarmOn)
				{
					CLR_BIT(PORTB, 1);
				}
				else
				{
					SET_BIT(PORTB, 1);
				}
				if(b_HeadsetOn)
				{
					RF_Transmit(2); //Turn on
					_delay_ms(1000);
					Message_Sent = 0;
				}
			}
			else
			{
				if((Message_Sent == 0) && (b_AlarmSounding == 0))
				{
					RF_Transmit(1); //Turn off
					_delay_ms(1000);
					Message_Sent = 1;
				}
				if(b_AlarmSounding && b_AlarmOn)
				{
					CLR_BIT(PORTB, 1);
				}
				else
				{
					SET_BIT(PORTB, 1);
				}
			}
		}
		else
			SET_BIT(PORTB, 1);
	}
}

void RF_Transmit(uint8_t data)
{
	int i=0;
	//This should be double the time of the data
	//to insure that we don't double check
	if(data == 1)
```

```
	{
		for(i=0;i<20;i++)
		{
			if(i%2 == 1)
			{
				CLR_BIT(PORTB, 0);
				_delay_ms(30);
			}
			else
			{
				SET_BIT(PORTB, 0);
				_delay_ms(15);
			}
		}
	}
	else if(data == 2)
	{
		for(i=0;i<40;i++)
		{
			if(i%2 == 1)
			{
				CLR_BIT(PORTB, 0);
				_delay_ms(30);
			}
			else
			{
				SET_BIT(PORTB, 0);
				_delay_ms(15);
			}
		}
	}
	else
	{
		CLR_BIT(PORTB, 0);
	}
}

void Init_Timers(void)
{
	//8 bit timer for lcd screen
	TCCR0 |= (0<<CS02)|(1<<CS01)|(0<<CS00);
	TIMSK|=(1<<TOIE0);
	TCNT0 = 0;

	//16 bit timer for increasing time
	TCCR1B|=(0<<CS02)|(1<<CS01)|(1<<CS00);
	TIMSK|=(1<<TOIE1);
	TCNT1= 65276; //49911;

	//8 bit timer for increasing seconds
	TCCR2 |= (1<<CS22)|(1<<CS21)|(1<<CS20);
	TIMSK|=(1<<TOIE2);
	TCNT2 = 0;
}

void Init_IO(void)
{
	DDRA = 0xFF;
	DDRB = 0x3F;
	DDRC = 0xFF;
	DDRD = 0x80;

	//Enable pull up resistors for buttons
	SET_BIT(PORTD, 6); //Set button
	SET_BIT(PORTD, 5); //Minute button
	SET_BIT(PORTD, 4); //Hour button
	SET_BIT(PORTD, 3); //Alarm button
```

```
        SET_BIT(PORTD, 2); //Snooze button
        SET_BIT(PORTD, 1); //IR input1 msb
        SET_BIT(PORTD, 0); //IR input2 lsb

        //enable pull up resistors for switches
        SET_BIT(PORTB, 6); //On/off Heaset
        SET_BIT(PORTB, 7); //On/off Alarm

        CLR_BIT(PORTB, 0); //For RF Transmit line
        SET_BIT(PORTB, 1); //For output to IR ucontroller for sound
}

ISR(TIMER2_OVF_vect)
{

        /*////////////////////////////////////////////////////////
        This is for the IR "interrupts" from the other ucontroller
        ////////////////////////////////////////////////////////*/
        if(GET_BIT(PIND, 0) == 0)
        {
                if(GET_BIT(PIND, 1) == 0)
                {
                        //This is the third button
                }
                else
                {
                        //This is the first button
                        if(b_AlarmSounding == 1)
                        {
                                b_AlarmSounding = 0;
                                return;
                        }
                }
        }

        else if(GET_BIT(PIND, 1) == 0)
        {
                //This is the second button
                if(b_AlarmSounding == 1)
                {
                        b_Snooze = 1;
                        Add_Snooze();
                        b_AlarmSounding = 0;
                        return;
                }
        }

        /*////////////////////////////////////////////////////////
        Settings for the alarm on/off and headset on/off
        ////////////////////////////////////////////////////////*/
        if(GET_BIT(PINB, 7) == 0)
        {
                //Sound the speaker
                b_AlarmOn = 1;

                if(GET_BIT(PINB, 6) == 0)
                {
                        //Sound the headset with the speaker
                        b_HeadsetOn = 1;
                }
                else
                {
                        b_HeadsetOn = 0;

                }

                if(!b_AlarmSounding)
```

```
        {
                SET_BIT(PORTB, 1);
        }
}
else if(GET_BIT(PINB, 6) == 0)
{
        //Sound just the headset, not the speaker
        b_HeadsetOn = 1;
        b_AlarmOn = 0;

        if(!b_AlarmSounding)
        {
                SET_BIT(PORTB, 1);
        }
}
else
{
        //Don't make any sounds, the alarm is off
        b_HeadsetOn = 0;
        b_AlarmOn = 0;
        SET_BIT(PORTB, 1);
}
/*////////////////////////////////////////////////////////
This sets the clock time
////////////////////////////////////////////////////////*/
if(GET_BIT(PIND, 6) == 0)
{
        b_SetAlarm = 0;
        if(GET_BIT(PIND, 5) == 0)
        {
                Increase_Mins();
        }
        else if(GET_BIT(PIND, 4) == 0)
        {
                Increase_Hrs();
        }
}
/*////////////////////////////////////////////////////////
This sets the alarm time
////////////////////////////////////////////////////////*/
else if(GET_BIT(PIND, 3) == 0)
{
        if(b_AlarmSounding == 1)
        {
                CLR_BIT(PORTB, 0);
                SET_BIT(PORTB, 1);
                b_AlarmSounding = 0;
                return;
        }
        b_SetAlarm = 1;

        if(GET_BIT(PIND, 5) == 0)
        {
                Increase_Alarm_Mins();
        }
        else if(GET_BIT(PIND, 4) == 0)
        {
                Increase_Alarm_Hrs();
        }
}
/*////////////////////////////////////////////////////////
This sets the snooze delay
////////////////////////////////////////////////////////*/
else if(GET_BIT(PIND, 2) == 0)
{
        if(b_AlarmSounding == 1)
        {
```

```
                        b_Snooze = 1;
                        Add_Snooze();
                        b_AlarmSounding = 0;
                        SET_BIT(PORTB, 1);
                        return;
                }

                b_SetSnooze = 1;
                if(GET_BIT(PIND, 5) == 0)
                {
                        Increase_Snooze_Mins();
                }
        }
        else
        {
                b_SetAlarm = 0;
                b_SetSnooze = 0;
        }
        TCNT2 = 0;
}
```

```
/*************************************************************
Travis Moore and Collin Barth

*The Discrete Digital Alarm Clock*
Electrical Engineering Senior Project
        Spring 2011
*************************************************************/
/*************************************************************
*LCD_Display.c*
This file controls all of the writing to the LCD Screen,
        clock functions, and the alarm methods
*************************************************************/
#include <avr/io.h>
#include <util/delay.h>
#include <bitmask.h>
#include <avr/interrupt.h>
#include <GlobalInclude.h>
#define F_CPU 1000000UL
/*************************************************************
* Initialization for variables for the LCD Screen
* -Define values for both sets of pins for each digit
*************************************************************/
//Define the digits for the lCD display
//Those with _1 use Vcc pin 15, _2 use Vcc pin 16
unsigned char digit1_2 = 0; //Pins C1, C2
unsigned char digit2_1 = 0; //Pins C2, C3, C4, C5
unsigned char digit2_2 = 0; //Pins C2, C3, C4, C5
unsigned char digit3_1 = 0; //Pins C6, C7 and A0, A1
unsigned char digit3_2 = 0; //Pins C6, C7 and A0, A1
unsigned char digit4_1 = 0; //Pins A1, A2, A3, A4
unsigned char digit4_2 = 0; //Pins A1, A2, A3, A4
/*************************************************************
* Declare some functions
*************************************************************/
void output_15(); //Pin A6
void output_16(unsigned char Minutes, unsigned char Hours); //Pin A7
void set_time(); //Increases the time if needed, sets the bits for the display
void Increase_Mins(void);
void Increase_Hrs(void);
void Increase_Alarm_Mins(void);
void Increase_Alarm_Hrs(void);
void Increase_Snooze_Mins(void);

int Check_Alarm(void);
short int Count_Clock = 0;
unsigned char Clear_Wait = 0;

void Increase_Mins(void)
{
        Minutes++;
        if(Minutes == 60)
        {
                Minutes = 0;
        }
        Seconds = 0;
}

void Increase_Alarm_Mins(void)
{
        Alarm_Minutes++;
        if(Alarm_Minutes == 60)
        {
                Alarm_Minutes = 0;
        }
}
```

```
void Increase_Hrs(void)
{
	Hours++;
	if(Hours == 12)
	{
		if(b_Clock_AM == 0)
		{
			b_Clock_AM = 1;
		}
		else
		{
			b_Clock_AM = 0;
		}
	}
	if(Hours == 13)
	{
		Hours = 1;
	}
	Seconds = 0;
}

void Increase_Alarm_Hrs(void)
{
	Alarm_Hours++;
	if(Alarm_Hours == 12)
	{
		if(b_Alarm_AM == 0)
		{
			b_Alarm_AM = 1;
		}
		else
		{
			b_Alarm_AM = 0;
		}
	}
	if(Alarm_Hours == 13)
	{
		Alarm_Hours = 1;
	}
}

void Add_Snooze(void)
{
	SET_BIT(PORTB, 1);

	Snooze_Alarm_Minutes = Minutes;
	Snooze_Alarm_Hours = Hours;
	Snooze_Alarm_Seconds = Seconds;

	b_Snooze_AM = b_Clock_AM;

	Snooze_Alarm_Minutes += Snooze_Minutes;

	if(Snooze_Alarm_Minutes >= 60)
	{
		Snooze_Alarm_Minutes -= 60;
		Snooze_Alarm_Hours++;

		if(Snooze_Alarm_Hours == 12)
		{
			if(b_Snooze_AM == 0)
			{
				b_Snooze_AM = 1;
			}
			else
			{
```

```
                                b_Snooze_AM = 0;
                        }
                }
                if(Snooze_Alarm_Hours == 13)
                {
                        Snooze_Alarm_Hours = 1;
                }
        }
        SET_BIT(PORTB, 1);
}

int Check_Alarm(void)
{
        if(b_SetAlarm == 0)
        {
                if(b_Snooze == 1)
                {
                        if(b_Snooze_AM == b_Clock_AM)
                        {
                                if(Snooze_Alarm_Seconds == Seconds)//if(Seconds <= 1)
                                {
                                        if(Snooze_Alarm_Hours == Hours)
                                        {
                                                if(Snooze_Alarm_Minutes == Minutes)
                                                {
                                                        b_AlarmSounding = 1;
                                                        b_Snooze = 0;
                                                        return 1;
                                                }
                                        }
                                }
                        }
                }
                else
                {
                        if(b_Alarm_AM == b_Clock_AM)
                        {
                                if(Seconds <= 1)
                                {
                                        if(Alarm_Hours == Hours)
                                        {
                                                if(Alarm_Minutes == Minutes)
                                                {
                                                        b_AlarmSounding = 1;
                                                        return 1;
                                                }
                                        }
                                }
                        }
                }
        }
        return 0;
}

void Increase_Snooze_Mins(void)
{
        Snooze_Minutes++;
        if(Snooze_Minutes >= 30)
        {
                Snooze_Minutes = 1;
        }
}

ISR(TIMER1_OVF_vect)
{
        Seconds++;
        if(Seconds == 60)
```

```
	{
		Minutes++;
		Seconds = 0;
	}
	if(Minutes == 60)
	{
		Hours++;
		Minutes = 0;
	}
	if((Hours == 12) && (Minutes == 0) && (Seconds == 0))
	{
		if(b_Clock_AM == 0)
		{
			b_Clock_AM = 1;
		}
		else
		{
			b_Clock_AM = 0;
		}
	}
	if(Hours == 13)
	{
		Hours = 1;
	}

	TCNT1 = 49911; //65276;  ;Reset timer value(2nd value is for 1s long minutes,
for quick testing)
}

ISR(TIMER0_OVF_vect)
{
	switch(Clear_Wait)
	{
		case 1:
			CLR_BIT(PORTA, 6);
			if(b_SetAlarm == 1)//Set_Alarm == 1)
			{
				//output_15();
				set_time(Alarm_Hours, Alarm_Minutes);
			}
			else if(b_SetSnooze == 1)
			{
				set_time(0, Snooze_Minutes);
			}
			else
			{
				set_time(Hours, Minutes);
			}
			output_15();
			break;
		case 5:
			output_15();
			break;
		case 6:
			SET_BIT(PORTA, 7);
			break;
		case 7:
			CLR_BIT(PORTA, 7);
			break;
		case 11:
			if(b_SetAlarm == 1)
			{
				output_16(Alarm_Minutes, Alarm_Hours);
			}
			else if(b_SetSnooze == 1)
			{
				output_16(Snooze_Minutes, 0);
```

```
                        }
                        else
                        {
                                output_16(Minutes, Hours);
                        }
                        break;
                case 12:
                        SET_BIT(PORTA, 6);
                        Clear_Wait = 0;
                        break;

                default:
                        Clear_Wait = Clear_Wait;
        }
        Clear_Wait++;
        TCNT0 = 120;
}

void set_time(int Hours, int Minutes)
{
        if(Hours >= 10)
        {
                digit1_2 = 3; //Pins 2 and 3
        }
        else
        {
                digit1_2 = 0;
        }
        switch(Hours%10)
        {
                case 1:
                        digit2_1 = 99;
                        digit2_2 = 6; //Pins 3,4,5,6
                        break;
                case 2:
                        digit2_1 = 14; //Pins 3,4,5,6
                        digit2_2 = 5; //Pins 3,4,5,6
                        break;
                case 3:
                        digit2_1 = 6; //Pins 3,4,5,6
                        digit2_2 = 7; //Pins 3,4,5,6
                        break;
                case 4:
                        digit2_1 = 5; //Pins 3,4,5,6
                        digit2_2 = 6; //Pins 3,4,5,6
                        break;
                case 5:
                        digit2_1 = 7; //Pins 3,4,5,6
                        digit2_2 = 3; //Pins 3,4,5,6
                        break;
                case 6:
                        digit2_1 = 15; //Pins 3,4,5,6
                        digit2_2 = 3; //Pins 3,4,5,6
                        break;
                case 7:
                        digit2_1 = 0; //Pins 3,4,5,6
                        digit2_2 = 7; //Pins 3,4,5,6
                        break;
                case 8:
                        digit2_1 = 15; //Pins 3,4,5,6
                        digit2_2 = 7; //Pins 3,4,5,6
                        break;
                case 9:
                        digit2_1 = 7; //Pins 3,4,5,6
                        digit2_2 = 7; //Pins 3,4,5,6
                        break;
                default:
```

```
                digit2_1 = 11; //Pins 3,4,5,6
                digit2_2 = 7;  //Pins 3,4,5,6
}
if(Minutes >= 50)
{
        digit3_1 = 10;
        digit3_2 = 14;
}
else if(Minutes >= 40)
{
        digit3_1 = 6;
        digit3_2 = 12;
}
else if(Minutes >= 30)
{
        digit3_1 = 14;
        digit3_2 = 6;
}
else if(Minutes >= 20)
{
        digit3_1 = 12;
        digit3_2 = 7;
}
else if(Minutes >= 10)
{
        digit3_1 = 6;
        digit3_2 = 0;
}
else
{
        digit3_1 = 14;
        digit3_2 = 11;
}
switch(Minutes%10)
{
        case 9:
                digit4_1 = 7;
                digit4_2 = 7;
                break;
        case 8:
                digit4_1 = 15;
                digit4_2 = 7;
                break;
        case 7:
                digit4_1 = 0;
                digit4_2 = 7;
                break;
        case 6:
                digit4_1 = 15;
                digit4_2 = 3;
                break;
        case 5:
                digit4_1 = 7;
                digit4_2 = 3;
                break;
        case 4:
                digit4_1 = 5;
                digit4_2 = 6;
                break;
        case 3:
                digit4_1 = 6;
                digit4_2 = 7;
                break;
        case 2:
                digit4_1 = 14;
                digit4_2 = 5;
                break;
```

```
                case 1:
                        digit4_1 = 0;
                        digit4_2 = 6;
                        break;
                default:
                        digit4_1 = 11;
                        digit4_2 = 7;
        }
}

void output_15()
{
        //Set the light for AM/PM
        if( (b_SetAlarm == 1) && b_Alarm_AM )
        {
                CLR_BIT(PORTB, 4);
        }
        else if( b_Clock_AM && (b_SetAlarm == 0) )
        {
                CLR_BIT(PORTB, 4);
        }
        else
        {
                SET_BIT(PORTB, 4);
        }
        //Set the light for alarm on
        if(b_AlarmOn)
        {
                SET_BIT(PORTC, 0);
        }
        else
        {
                CLR_BIT(PORTC, 0);
        }
        //Clear the light for headset on
        if(b_HeadsetOn)
        {
                SET_BIT(PORTC, 1);
        }
        else
        {
                //For digit 1
                CLR_BIT(PORTC, 1);
        }

        SET_BIT(PORTA, 5); //For the Colon

        switch(digit2_1)
        {
                case 15:
                        SET_BIT(PORTC, 2);
                        SET_BIT(PORTC, 3);
                        SET_BIT(PORTC, 4);
                        SET_BIT(PORTC, 5);
                        break;
                case 14:
                        SET_BIT(PORTC, 2);
                        SET_BIT(PORTC, 3);
                        SET_BIT(PORTC, 4);
                        CLR_BIT(PORTC, 5);
                        break;
                case 11:
                        SET_BIT(PORTC, 2);
                        CLR_BIT(PORTC, 3);
                        SET_BIT(PORTC, 4);
                        SET_BIT(PORTC, 5);
                        break;
```

```
        case 7:
                CLR_BIT(PORTC, 2);
                SET_BIT(PORTC, 3);
                SET_BIT(PORTC, 4);
                SET_BIT(PORTC, 5);
                break;
        case 6:
                CLR_BIT(PORTC, 2);
                SET_BIT(PORTC, 3);
                SET_BIT(PORTC, 4);
                CLR_BIT(PORTC, 5);
                break;
        case 5:
                CLR_BIT(PORTC, 2);
                SET_BIT(PORTC, 3);
                CLR_BIT(PORTC, 4);
                SET_BIT(PORTC, 5);
                break;
        default:
                CLR_BIT(PORTC, 2);
                CLR_BIT(PORTC, 3);
                CLR_BIT(PORTC, 4);
                CLR_BIT(PORTC, 5);
}
//SET THE MINUTE BITS (10s place)
switch(digit3_1)
{
        case 14:
                SET_BIT(PORTC, 6);
                SET_BIT(PORTC, 7);
                SET_BIT(PORTA, 0);
                CLR_BIT(PORTA, 1);
                break;
        case 12:
                SET_BIT(PORTC, 6);
                SET_BIT(PORTC, 7);
                CLR_BIT(PORTA, 0);
                CLR_BIT(PORTA, 1);
                if(digit4_1 == 14 || digit4_1 == 15 || digit4_1 == 11)
                {
                        SET_BIT(PORTA, 1);
                }
                else
                {
                        CLR_BIT(PORTA, 1);
                }
                break;
        case 10:
                SET_BIT(PORTC, 6);
                CLR_BIT(PORTC, 7);
                SET_BIT(PORTA, 0);
                CLR_BIT(PORTA, 1);
                if(digit4_1 == 14 || digit4_1 == 15 || digit4_1 == 11)
                {
                        SET_BIT(PORTA, 1);
                }
                else
                {
                        CLR_BIT(PORTA, 1);
                }
                break;
        case 6:
                CLR_BIT(PORTC, 6);
                SET_BIT(PORTC, 7);
                SET_BIT(PORTA, 0);
                CLR_BIT(PORTA, 1);
                break;
```

```
		default:
			CLR_BIT(PORTC, 6);
			CLR_BIT(PORTC, 7);
			CLR_BIT(PORTA, 0);
			CLR_BIT(PORTA, 1);
	}

	//SET THE MINUTE BITS (1s place)
	switch(digit4_1)
	{
		case 15:
			SET_BIT(PORTA, 1);
			SET_BIT(PORTA, 2);
			SET_BIT(PORTA, 3);
			SET_BIT(PORTA, 4);
			break;
		case 14:
			SET_BIT(PORTA, 1);
			SET_BIT(PORTA, 2);
			SET_BIT(PORTA, 3);
			CLR_BIT(PORTA, 4);
			break;
		case 11:
			SET_BIT(PORTA, 1);
			CLR_BIT(PORTA, 2);
			SET_BIT(PORTA, 3);
			SET_BIT(PORTA, 4);
			break;
		case 7:
			CLR_BIT(PORTA, 1);
			SET_BIT(PORTA, 2);
			SET_BIT(PORTA, 3);
			SET_BIT(PORTA, 4);
			break;
		case 6:
			CLR_BIT(PORTA, 1);
			SET_BIT(PORTA, 2);
			SET_BIT(PORTA, 3);
			CLR_BIT(PORTA, 4);
			break;
		case 5:
			CLR_BIT(PORTA, 1);
			SET_BIT(PORTA, 2);
			CLR_BIT(PORTA, 3);
			SET_BIT(PORTA, 4);
			break;
		default:
			CLR_BIT(PORTA, 1);
			CLR_BIT(PORTA, 2);
			CLR_BIT(PORTA, 3);
			CLR_BIT(PORTA, 4);
	}
}


void output_16(unsigned char Minutes, unsigned char Hours)
{
	//For AM/PM Indicator
	CLR_BIT(PORTB, 4);

	//Clr the light for alarm on
	CLR_BIT(PORTC, 0);

	switch(digit1_2)
	{
		case 3:
```

```
                SET_BIT(PORTC, 1);
                SET_BIT(PORTC, 2);
        break;
        default:
                CLR_BIT(PORTC, 1);
}
switch(digit2_2)
{
        case 7:
                if(Hours >= 10)
                {
                        SET_BIT(PORTC, 2);
                }
                else
                {
                        CLR_BIT(PORTC, 2);
                }
                SET_BIT(PORTC, 3);
                SET_BIT(PORTC, 4);
                SET_BIT(PORTC, 5);
                break;
        case 6:
                if(Hours >= 10)
                {
                        SET_BIT(PORTC, 2);
                }
                else
                {
                        CLR_BIT(PORTC, 2);
                }
                SET_BIT(PORTC, 3);
                SET_BIT(PORTC, 4);
                CLR_BIT(PORTC, 5);
                break;
        case 5:
                if(Hours >= 10)
                {
                        SET_BIT(PORTC, 2);
                }
                else
                {
                        CLR_BIT(PORTC, 2);
                }
                SET_BIT(PORTC, 3);
                CLR_BIT(PORTC, 4);
                SET_BIT(PORTC, 5);
                break;
        case 3:
                if(Hours >= 10)
                {
                        SET_BIT(PORTC, 2);
                }
                else
                {
                        CLR_BIT(PORTC, 2);
                }
                CLR_BIT(PORTC, 3);
                SET_BIT(PORTC, 4);
                SET_BIT(PORTC, 5);
                break;
        default:
                if(Hours >= 10)
                {
                        SET_BIT(PORTC, 2);
                }
                else
                {
```

```
                        CLR_BIT(PORTC, 2);
                }
                CLR_BIT(PORTC, 3);
                CLR_BIT(PORTC, 4);
                CLR_BIT(PORTC, 5);
        }
        switch(digit3_2)
        {
                case 15:
                        SET_BIT(PORTC, 6);
                        SET_BIT(PORTC, 7);
                        SET_BIT(PORTA, 0);
                        SET_BIT(PORTA, 1);
                        break;
                case 14:
                        SET_BIT(PORTC, 6);
                        SET_BIT(PORTC, 7);
                        SET_BIT(PORTA, 0);
                        CLR_BIT(PORTA, 1);
                        break;
                case 12:
                        SET_BIT(PORTC, 6);
                        SET_BIT(PORTC, 7);
                        CLR_BIT(PORTA, 0);
                        CLR_BIT(PORTA, 1);
                        break;
                case 11:
                        SET_BIT(PORTC, 6);
                        CLR_BIT(PORTC, 7);
                        SET_BIT(PORTA, 0);
                        SET_BIT(PORTA, 1);
                        break;
                case 7:
                        CLR_BIT(PORTC, 6);
                        SET_BIT(PORTC, 7);
                        SET_BIT(PORTA, 0);
                        SET_BIT(PORTA, 1);
                        break;
                case 6:
                        CLR_BIT(PORTC, 6);
                        SET_BIT(PORTC, 7);
                        SET_BIT(PORTA, 0);
                        CLR_BIT(PORTA, 1);
                        break;
                case 0:
                        CLR_BIT(PORTC, 6);
                        CLR_BIT(PORTC, 7);
                        CLR_BIT(PORTA, 0);
                        CLR_BIT(PORTA, 1);
                        break;
                default:
                        CLR_BIT(PORTC, 6);
                        CLR_BIT(PORTC, 7);
                        CLR_BIT(PORTA, 0);
                        CLR_BIT(PORTA, 1);
        }
        switch(digit4_2)
        {
                case 7:
                        CLR_BIT(PORTA, 1);
                        SET_BIT(PORTA, 2);
                        SET_BIT(PORTA, 3);
                        SET_BIT(PORTA, 4);
                        if(Minutes == 0 || Minutes == 3 || Minutes == 7 || Minutes == 8
|| Minutes == 9 || Minutes == 20 ||
                                Minutes == 23 || Minutes == 27 || Minutes == 28  ||
Minutes == 29)
```

```
                        {
                                SET_BIT(PORTA, 1);
                        }
                        break;
                case 6:
                        CLR_BIT(PORTA, 1);
                        SET_BIT(PORTA, 2);
                        SET_BIT(PORTA, 3);
                        CLR_BIT(PORTA, 4);
                        if(Minutes == 1 || Minutes == 4 || Minutes == 21 || Minutes ==
24)
                        {
                                SET_BIT(PORTA, 1);
                        }
                        break;
                case 5:
                        CLR_BIT(PORTA, 1);
                        SET_BIT(PORTA, 2);
                        CLR_BIT(PORTA, 3);
                        SET_BIT(PORTA, 4);
                        if(Minutes == 2 || Minutes == 22)
                        {
                                SET_BIT(PORTA, 1);
                        }
                        break;
                case 3:
                        CLR_BIT(PORTA, 1);
                        CLR_BIT(PORTA, 2);
                        SET_BIT(PORTA, 3);
                        SET_BIT(PORTA, 4);
                        if(Minutes == 5 || Minutes == 6 || Minutes == 25 || Minutes ==
26)
                        {
                                SET_BIT(PORTA, 1);
                        }
                        break;
                case 0:
                        CLR_BIT(PORTA, 1);
                        CLR_BIT(PORTA, 2);
                        CLR_BIT(PORTA, 3);
                        CLR_BIT(PORTA, 4);
                        break;
                default:
                        CLR_BIT(PORTA, 1);
                        CLR_BIT(PORTA, 2);
                        CLR_BIT(PORTA, 3);
                        CLR_BIT(PORTA, 4);
        }
}
```

```
/*************************************************************
Travis Moore and Collin Barth

*The Discrete Digital Alarm Clock*
Electrical Engineering Senior Project
        Spring 2011
*************************************************************/

/*************************************************************
*IR_Receiver.c*
This file contains the code controlling second internal
        microcontroller, which hadles the IR signal reception
         as well as the main clock's speaker function
*************************************************************/
#include <avr/io.h>
#include <util/delay.h>
#include <bitmask.h>
#include <avr/interrupt.h>

#ifndef F_CPU
//define cpu clock speed if not defined
#define F_CPU 1000000UL
#endif

void IR_Init(void);
void Check_IR(void);

int main(void)
{
        IR_Init();

        while(1)
        {
                if(GET_BIT(PIND, 6) == 0)
                {
                        _delay_ms(4);
                        if(GET_BIT(PIND, 6) == 0)
                        {
                                Check_IR();
                        }
                }
                else
                {
                        CLR_BIT(PORTD, 7);
                }
        }
}

void IR_Init(void)
{
        DDRD = 0xB0;
        SET_BIT(PORTD, 6); //enable pull up resistor on port 6 , the input
}

void Check_IR(void)
{
        _delay_us(38000);

        //SET_BIT(PORTD, 7);
        //_delay_ms(1);
        if(GET_BIT(PIND, 6) == 0)
        {
                //CLR_BIT(PORTD, 7);
                //Button pressed is either 2 or 4
                _delay_us(3400);

                if(GET_BIT(PIND, 6) == 0)
```

```
                {
                        //Button is 4
                        SET_BIT(PORTD, 5);
                        //_delay_ms(1);
                }
                else
                {
                        //Button is 2
                        SET_BIT(PORTD, 4);
                        //_delay_ms(1);
                }
        }
        //Button pressed must be either 1 or 3
        else
        {
                _delay_us(3250);
                if(GET_BIT(PIND, 6) == 0)
                {
                        //Button is 3
                        CLR_BIT(PORTD, 5);
                }
                else
                {
                        //Button is 1
                        CLR_BIT(PORTD, 4);
                }
        }
}
```

```
/*************************************************************
Travis Moore and Collin Barth

*The Discrete Digital Alarm Clock*
Electrical Engineering Senior Project
        Spring 2011
*************************************************************/

/*************************************************************
*RF_Receiver_32.c*
This file controls the RF receiver chip. It figures out the
        incoming RF codes, removes the noise, and filters out the
        necessary signals that we are looking for
*************************************************************/

#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>
#include <bitmask.h>

#ifndef F_CPU
//define cpu clock speed if not defined
#define F_CPU 1000000UL
#endif

unsigned char DetectSync();
void Main_Init(void);
unsigned char CheckHigh(void);
unsigned char CheckLow(void);
void SoundAlarm(void);

unsigned char Alarm_Sounding = 0;
unsigned char FirstCheck(void);

int Alarm(void);
int check = 0;

int main(void)
{
        Main_Init();
        while(1)
        {
                int result = 0;
                result = DetectSync();

                if(result == 1)
                {
                        SET_BIT(PORTB, 2);
                        CLR_BIT(PORTB, 1);
                        _delay_ms(500);
                        Alarm_Sounding = 1;
                }
                else if(result == 2)
                {
                        Alarm_Sounding = 0;
                        SET_BIT(PORTB, 1);
                        CLR_BIT(PORTB, 2);
                        CLR_BIT(PORTB, 0);
                        _delay_ms(500);
                }
                else
                {
                        CLR_BIT(PORTB, 2);
                        CLR_BIT(PORTB, 1);
                }
```

```
                if(Alarm_Sounding == 1)
                {
                        check = Alarm();
                        if(check > 0)
                        {
                                if(check == 2)
                                {
                                        Alarm_Sounding = 0;
                                        SET_BIT(PORTB, 1);
                                        CLR_BIT(PORTB, 2);

                                        CLR_BIT(PORTB, 0);
                                        _delay_ms(2000);
                                }
                        }
                }
        }
        return 0;
}

int Alarm(void)
{
        int result = 0;
        CLR_BIT(PORTC, 1);
                result = DetectSync();
                if(result == 0)
                {
                        for(int j=0; j<30; j++)
                        {
                                result = DetectSync();
                                if(result == 0)
                                {
                                        SET_BIT(PORTC, 0);
                                        _delay_ms(1);
                                }
                                else
                                {
                                        j=30;
                                        SET_BIT(PORTC, 1);
                                        CLR_BIT(PORTC, 0);
                                        return result;

                                }
                        }
                        for(int j=0; j<30; j++) //Outputs Buzzer soind instead of
constant noise
                        {
                                result = DetectSync();
                                if(result == 0)
                                {
                                        CLR_BIT(PORTC, 0);
                                        _delay_ms(1);
                                }
                                else
                                {
                                        j=30;
                                        SET_BIT(PORTC, 1);
                                        return result;
                                }
                        }
                }
                else
                {
                        SET_BIT(PORTC, 1);
                        CLR_BIT(PORTC, 0);
                        return result;
                }
```

```
        return 0;
}
unsigned char CheckHigh(void)
{
        if(GET_BIT(PIND, 5) == 0)
        {
                return 0;
        }
        else
                return 1;
}

unsigned char CheckLow(void)
{
        if(check == 0)
        {
                SET_BIT(PORTC, 2);
                check = 1;
        }
        else
        {
                CLR_BIT(PORTC, 2);
                check = 0;
        }
        if(GET_BIT(PIND, 5) == 0)
        {
                SET_BIT(PORTA, 1);
                return 1;
        }
        else
        {
                CLR_BIT(PORTA, 1);
                return 0;
        }
}

unsigned char FirstCheck(void)
{
        if(CheckLow()) {
                _delay_ms(2);
                if(CheckLow()) {
                        _delay_ms(2);
                                if(CheckLow()) {
                                        _delay_ms(2);
                                        if(CheckLow()) {
                                                _delay_ms(2);
                                                if(CheckLow()) {
                                                        _delay_ms(2);
                                                        if(CheckLow()) {
                                                                _delay_ms(2);
                                                                if(CheckLow()) {
                                                                        _delay_ms(2);
                                                                        if(CheckLow()) {


        _delay_ms(2);

        if(CheckLow()) {

        _delay_ms(2);

        if(CheckLow()){

        _delay_ms(2);
                //20 ms time elapsed
                                        return 1;
                                                                                }
                                                                                }else
```

```
        return 0;
                                                                        }else
                                                                                return 0;
                                                                }else
                                                                        return 0;
                                                        }else
                                                                return 0;
                                                }else
                                                        return 0;
                                        }else
                                                return 0;
                                }else
                                        return 0;
                        }else
                                return 0;
                }else
                        return 0;
}


//This function will return a 1 if we receive a complete Sync byte
unsigned char DetectSync()
{
        unsigned char average = 0;
        unsigned char averageSTOP = 0;

        if(FirstCheck() == 0)
        {
                return 0;
        }
        if(CheckLow())
        {
                _delay_ms(17);
        }
        else
        {
                return 0;
        }
        if(!(CheckLow()))
        {
                _delay_ms(2);
                if(!(CheckLow()))
                {
                        _delay_ms(2);
                        if(!(CheckLow()))
                        {
                                _delay_ms(19);
                                if(CheckLow())
                                {
                                        _delay_ms(4);
                                        if(CheckLow())
                                        {
                                                _delay_ms(4);
                                                if(CheckLow())
                                                {
                                                        _delay_ms(4);
                                                        if(CheckLow())
                                                        {
                                                                while(CheckLow());
                                                        }
                                                        else{return 0;}
                                                }
                                                else{return 0;}
                                        }
                                        else{return 0;}
                                }
```

```
                        else{return 0;}
                }
                else{return 0;}
        }
        else{return 0;}

        _delay_us(8800);

        for(int i=0;i<9;i++)
        {
                average += (CheckHigh());
                _delay_us(24800);
                average += (CheckLow());
                _delay_ms(3);
                average += (CheckLow());
                _delay_us(24800);
        }
        _delay_ms(25);
        for(int i=0; i<50; i++)
        {
                averageSTOP += (CheckHigh());
                _delay_ms(1);
        }
        if(averageSTOP>45)
        {
                return 2;
        }
        else if(average>15)
        {
                return 1;
        }
        else
        {
                return 0;
        }
}

void Main_Init(void)
{
        //define port A,B,C as output
        DDRA=0xFF;
        DDRB=0xFF;
        DDRC=0xFF;
        //define port D as input
        DDRD=0x00;

        //Port D, 5 will be used as the Rx input, disable pullup resistor
        SET_BIT(PORTD, 5);

        //Make sure speaker is off
        CLR_BIT(PORTB, 0);

        //LED for Alarm OFF
        CLR_BIT(PORTB, 1);

        //LED for Alarm ON
        CLR_BIT(PORTB, 2);
}
```